uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería de Sistemas de Comunicaciones
2017-2018

Trabajo Fin de Grado

"Redes profundas para la detección de vehículos potencialmente peligrosos en Smartcities"

Rocío Gómez-Choco González

Tutor

Harold Molina-Bulla

Agradecimientos

Agradecer en primer lugar a mi tutor, Harold, por la ayuda que me ha brindado durante los meses de desarrollo del proyecto. Él ha contribuido a que conozca un mundo nuevo, el gran conocido Big Data.

Y no podía faltar mi familia, ellos han sido mi vía de escape para los momentos más duros. Los que me animaban y me animan para luchar ante las dificultades. Los que hacen que un mal día se convierta en bueno.

Resumen

El mundo actual cuenta con un fenómeno llamado *Big Data*, que surgió de la generación de ingentes cantidades de datos. A partir de ellos es posible obtener todo tipo de información y, en este trabajo, se busca hacer un buen uso de ellos aplicándolo a la seguridad ciudadana.

Para tratar estos datos se cuenta con mecanismos como *Machine Learning*, o aprendizaje máquina, y el caso específico denominado *Deep Learning*, o aprendizaje profundo. Las redes profundas son un tipo de redes neuronales que cuando intentan aprender algo, lo hacen de la misma manera que lo haría un cerebro humano. Para ello, se requiere entrenar estas redes con una serie de librerías, bases de datos y programas determinados con el fin de aprovechar sus ventajas para obtener un beneficio para los ciudadanos. Este último es el que se va a emplear en este proyecto

El mundo de las redes neuronales proporciona un sinfín de posibilidades de aplicación y cada vez más sofisticadas. Ayudan a realizar trabajos de manera más eficiente, más rápida o a proporcionar información interesante sobre aquellos elementos que analizan, como imágenes para el reconocimiento de enfermedades, reconocimiento de sentimientos, entre otras.

En este proyecto se aplican las redes profundas a la seguridad ciudadana con un método de vigilancia pasiva. En él se analizan las imágenes capturadas por las cámaras de las calles a supervisar, con el objetivo de detectar un vehículo de gran tamaño y generar una alarma. De esta manera la zona quedaría vigilada y se podrían evitar posibles altercados. Esta puede ser una contribución más para tener una ciudad inteligente.

Las *smart cities*, o ciudades inteligentes, son aquellas ciudades interconectadas mediante las nuevas tecnologías. Esto provoca un desarrollo sostenible, aumento de la calidad de vida, eficacia de los recursos disponibles y participación de la ciudadanía.

Es crucial el papel que juegan los sensores inteligentes, ya que ellos son los que informan de las mediciones de los datos que recogen. Miden niveles de radiación, llenado de cubos de basura, aparcamientos libres, detección de señal wifi, entre otras.

Algunas ciudades que ya aplican este modelo son Santander, Londres o Tokio. Son extensos los ámbitos de aplicación, entre ellos se encuentran:

- Medio ambiente: con medidas como fomentar el reciclaje.
- Sanidad: cuenta con teleasistencia.
- Urbanismo: dispone de nuevos sistemas de alumbrado LED.
- La Administración: tiene plataformas de pago online o wifi público gratuito.
- Seguridad: dispone de los efectivos de emergencia como el Samur o la policía con tiempos de respuesta reducidos.
- Turismo: con aplicaciones turísticas y guías de ocio adaptadas en función de los gustos del usuario.

Por tanto, para llevar a cabo la implementación de este sistema en la vida real, es necesaria su ejecución y realizar un posterior análisis de los resultados para evaluar los dos factores más importantes: el tiempo de respuesta que requiere la red para proporcionar resultados y la precisión de dichos resultados.

El tiempo de respuesta para generar la alarma no será en tiempo real, que sería lo más deseable. En cuanto a la precisión de acierto en los resultados, un sistema no debería darse por válido si no supera el 75 % de precisión. Pero esto depende del problema a evaluar y en función de eso, debe ajustarse debidamente. En este estudio, se han obtenido resultados de un 95 % de precisión de acierto en la clasificación de test en media, por lo que se ha considerado que es una buena solución.

Para proceder con la ejecución de esta red neuronal, en primer lugar, ha sido necesario realizar un estudio de dichas redes: cómo surgieron, cómo se estructuran, qué elementos tienen, cómo trabajar con ellas, los tipos de redes neuronales y qué tipo en concreto es el necesario en este estudio.

A pesar de encontrar dos tipos redes neuronales, al realizar una comparación entre ellas, es fácil elegir la que se adapta a este proyecto. Se encuentran las redes neuronales recurrentes (RNN, *recurrent neural network*) y las redes neuronales convolucionales (CNN, *convolutional neural network*). La diferencia entre ellas que hace tan fácil la decisión por una de ellas, es que las RNN son ideales para el análisis de texto y voz,

mientras que las CNN, son ideales para el procesamiento de imágenes y video. Por este motivo elegimos las CNN.

Una vez está elegido el tipo de red que se va a emplear, es necesario saber qué tipo de librería se puede utilizar para lograr la finalidad buscada. Resulta sencillo encontrar una gran cantidad de librerías comúnmente usadas en estos entornos de trabajo. Entre ellas están: *Caffe, Theano* o *Tensorflow*, entre otras. Resulta necesaria la comparación entre estas tres, para poder usar la que resulte más eficaz. Una vez valoradas sus ventajas y desventajas y analizando cuál de ellas se puede adaptar más a los dos requisitos primordiales que son necesarios, tiempo de respuesta y precisión que proporcionan, se decide usar *Tensorfllow*.

Todo lo explicado hasta ahora, se adapta a un problema de ejecución en una sola máquina, pero en entornos *Big Data* (o Mega Datos (RAE)) se necesitan soluciones orientadas a estos entornos, es decir, se necesitan soluciones adaptadas para procesar grandes cantidades de datos. Por lo que es necesario realizar un procesamiento distribuido de los datos. Las dos plataformas más comunes para llevar esto a cabo son *Hadoop* y *Spark*. Tras realizar una comparativa entre ellas, se decide usar *Spark* debido a su flexibilidad y velocidad.

Estas dos elecciones anteriores, se unifican en una única solución gracias a *Tensorflow on Spark*, una librería que permite el entrenamiento distribuido de *Tensorflow* y la inferencia de *Apache Spark*. También existen *Theano on Spark* y *Caffe on Spark*. Pero no han sido elegidas como herramientas a utilizar.

Tensorflow on Spark cuenta con múltiples utilidades y ventajas para estos entornos de trabajo, algunas son: busca minimizar la cantidad de cambios de código necesarios para ejecutar programas existentes, admite funcionalidades del *Tensorflow* original tales como la inferencia o el uso de *tensorboard*.

Las ejecuciones han sido llevadas a cabo en los servidores disponibles en la universidad, pero el sistema se ha desarrollado de forma transparente para poder implementarlo en cualquier *cluster*, como podría haber sido AWS (*Amazon Web Services*).

El conjunto de imágenes que han sido usadas, es CIFAR-10, que cuenta con 60000 imágenes y, entre ellas, diez categorías distintas de objetos. Además, se han separado en dos bases de datos distintas:

- Conjunto de entrenamiento, que cuenta con 50000 imágenes.
- Conjunto de test, que cuenta con el resto de imágenes, 10000.

Resulta muy importante que ambos conjuntos no tengan los mismos datos, ya que esto podría provocar un sobreajuste de los datos. El sobreajuste consiste, a grandes rasgos, en que el sistema aprende demasiado bien los datos de entrenamiento que le son proporcionados. Esto lo que provoca es, que al proporcionarle datos distintos, en este caso serían los datos de evaluación, se obtendrían muy malos resultados de precisión en el sistema.

El sobreajuste es un fallo muy común que se produce en las redes neuronales, pero con cuidado y ajustando los parámetros es posible evitarlo o, al menos, reducirlo de manera que proporcione unos resultados aceptables.

Ya que CIFAR-10 cuenta con diez categorías de imágenes, la salida de la red neuronal proporcionará diez clasificaciones distintas. Pero este no es el objetivo del estudio. El objetivo es que el sistema pueda clasificar en solo tres categorías:

- Detección de coche
- Detección de camión
- Ninguna de las anteriores

Así, se obtiene un resultado adaptado al problema a resolver.

Para lograr conseguir esta adaptación, es necesario hacer una modificación en la parte del código de la red neuronal en la que se hace la carga de las imágenes. En concreto, en la parte donde se realiza el etiquetado de cada imagen. De esta manera, queda modificado el código original para poder cumplir las necesidades de este estudio.

Una vez quedan claros los procesos a seguir, se ha realizado en primer lugar el caso de la ejecución en una única máquina. Este caso no es que aporte novedad ni sea el objetivo que pretende conseguir este estudio, pero resulta necesario para realizar una primera toma de contacto con los códigos, las ejecuciones y las máquinas empleadas.

Por este motivo, y sin tener en cuenta la modificación que se va a realizar en las salidas, se procede a la ejecución del problema con el código original y teniendo la clasificación de las diez categorías. Este es el primer paso tomado en cuanto a ejecuciones para adquirir soltura.

Una vez se han comprendido los distintos factores que rodean a este ejemplo, tales como problemas surgidos con la comunicación con los servidores o problemas de comprensión de los resultados, se procede a avanzar un paso más.

El siguiente paso dado ha sido realizar la ejecución del caso en el que las salidas han sido modificadas, es decir, la clasificación de los tres posibles casos que queremos obtener. Para realizar una mejor compresión de los resultados, se ha ejecutado una serie de diez lanzamientos distintos para poder analizar los resultados de cada uno de ellos.

Realizando una media de los diez lanzamientos, se obtiene un tiempo de ejecución de 62 minutos y 52 segundos con una varianza de 0,00125, y una precisión de acierto en el test del 95 % con una varianza de 0,0000004.

Una vez tenemos estos resultados, es momento de evaluar dichas ejecuciones pero en un entorno distribuido. Esto es necesario ya que en entornos *Big Data*, la ejecución en una sola máquina no suele llevarse a la práctica pues el sistema queda restringido a las limitaciones de dicha máquina, que serían su capacidad de almacenamiento y sus limitaciones de procesamiento.

Sin embargo, es necesario destacar que al realizar ejecuciones en paralelo el sistema puede degradarse debido a que la suma de los tiempos de ejecución de todas las máquinas empleadas puede ser mayor que el tiempo de ejecución en una sola máquina, debido a factores propios de cada máquina como el intercambio de mensajes de información. Esto dependerá del problema y los datos tratados.

Para la ejecución distribuida se tiene un entorno *Hadoop*, *YARN* y *Spark* que cuenta con una estructura en la que están presenten nodos con distintas funcionalidades:

- Cliente y el *Resource Manager*, que son los encargados de decidir qué nodo será el *driver* y qué nodos los esclavos (la decisión se toma al azar).

- *Driver*: analiza, distribuye y programa el trabajo que es necesario realizar con los datos que tiene cada problema en concreto.
- Nodos de cómputo o *executors*: son los nodos que llevan a cabo el procesamiento de los datos que le son asignados. Deben informar del estado de procesamiento al *driver*.

Los resultados obtenidos para esta ejecución han sido una precisión de acierto media del 86,8% con una varianza de 0,000002 y un tiempo medio de entrenamiento de 2 horas y 43 minutos, con una varianza de 0,0023.

Si se comparan los dos tipos de ejecuciones se observa que el procesamiento lineal tiene una mayor precisión de acierto en el test que el caso distribuido. Sin embargo, este último caso mejora el manejo de los datos, lo cual resulta muy interesante para los temas de *Big Data*.

Por tanto, se elige una ejecución distribuida en aquellos casos en los que realmente la cantidad de datos a manejar es enorme. Para el resto de casos, no sería la mejor opción a escoger.

Abstract

The current world has a phenomenon called Big Data, that emerged from the generation of huge amounts of data. From them it is possible to obtain all kinds of information and, in this project, we seek to make good use of them by applying it to citizen security.

To deal with these data there are mechanisms like Machine Learning and the specific case called Deep Learning. Deep networks are a type of neural networks that when they try to learn somethin, they do it in the same way that a human brain would. To do this, we need to train these networks with libraries, databases and specific programs in order to take advantage of their benefits for citizens. The latter is the one that will be used in this project.

The world of neural networks provides endless possibilities of application and increasingly sophisticated. They help to carry out work more efficiently, faster or to provide interesting information about the elements they analyze, such as images for the recognition of diseases, recognition of feelings, among others.

In this project, Deep networks are applied to citizen security with a passive surveillance method. It analyzes the images captured by the cameras on the streets to be monitored, in order to detect a large vehicle and generate an alarm. In this way, the área would be monitored and possible altercations could be avoided. This can be a contribution more to have a smart city.

Smart cities, are those cities interconnected through new technologies. This causes a sustainable development, an increase in the quality of life, effectiveness of available resources and participation of the citizens.

The role played by intelligent sensors is crucial, since they are the ones that inform the measurements of the data they collect. Measure radiation levels, fill garbage bins, free parking, wifi signal detection, among others.

Some cities that already apply this model are Santander, London or Tokyo. The áreas of application are extensive, including:

- Environment: with measures such as promoting recycling.

- Health care: with telecare.
- Urbanism: has the new LED lighting systems.
- The Administration: has online payment platforms or free public WI-FI.
- Security: has emergency personnel such as the Samur or the police with reduced response times.
- Tourism: with tourist applications and leisure guides adapted according to the user's preferences.

Therefore, to carry out the implementation of this system in real life, it is necessary to implement and carry out a subsequent analysis of the results to evaluate the two most important factors: the response time required by the network to provide results and the precision of those results.

The response time to generate the alarm will not be in real time, which would be the most desirable. As for the precision of accuracy in the results, a system should not be considered valid if it does not exceed 75 % accuracy. But his depends on the problem to be evaluated and based on that, it must be adjusted properly. In this project, results have been obtained with 95 % accuracy on average test classification, which is why it has been considered a good solution.

To proceed with the execution of this neural network, first of all, it has been necessary to carry out a study of these networks: how they arose, how they are structured, what elements they have, how to work with them, the types of neural networks and what type in particular is the one needed in this project.

Despite finding two types of neural networks, when making a comparison between them, it is easy to choose the one that adapts to this project. Recurrent neuronal networks (RNN) and convolutional neural networks (CNN) are found. The difference between them that makes the decision so easy for one of them, is that the RNN are ideal for the analysis of text and voice, while the CNN, are ideal for the processing of images and video. For this reason we chose CNN.

Once the type of network that will be used is chosen, it is necessary to know what type of library can be used to achieve the desired purpose. It is easy to find a large number of libraries commonly used in these work environments. Among them are:

Caffe, Theano or Tensorflow, among others. It is necessary to compare these three, in order to use the one that is most effective. Once their advantages and disadvantages have been evaluated and analyzing which of them can be adapted more to the two fundamental requirements that are necessary, response time and precisión they provide, it is decided to use Tensorflow.

Everything explained up to now, adapts to an execution problem in a single machine, but in Big Data environments (or Mega Data (RAE)) solutions are needed oriented to these environments, that is, adapted solutions are needed for large quantities of data. Therefore it is necessary to perform a distributed processing of the data. The two most common platforms to carry this are Hadoop and Spark. After making a comparison between them, decide to use Spark due to its flexibility and speed.

These two previous elections are unified in a single solution thanks to Tensorflow on Spark, a library that allows the distributed training of Tensorflow and the inference of Apache Spark. There are also Theano on Spark and Caffe on Spark. But they have not been chosen as tools to use.

Tensorflow on Spark has multiple utilities and advantages for these work environments, some are: it seeks to minimize the amount of code changes necessary to run existing programs, it supports original Tensorflow functionalities such as inference or the use of tensorboard.

The executions have been carried out in the available servers in the university, but the system has been developed in a transparent way to be able to implement it in any cluster, as it could have been AWS (Amazon Web Services).

The set of images that have been used is CIFAR-10, which has 60000 images and, among them, ten different categories of objects. In addition, they have been separated into two different databases:

- Training set, which has 50000 images.
- Test set, which has the rest of the images, 10000.

It is very important that both sets do not have the same data, since this could cause an overfitting of the data. The overfitting consists, roughly, in that the system

learns too well the training data that are provided to it. This is what causes, that by providing different data, in this case would be the evaluation data, very poor results of accuracy in the system would be obtained.

Overfitting is a very common failure that occurs in neural networks, but with care and adjusting the parameters it is possible to avoid it or, at least, reduce it in a way that provides acceptable results.

Since CIFAR-10 has ten categories of images, the output of the neural network will provide ten different classifications. But this is not the objective of the project. The objective is that the system can be classified into only three categories:

- Car detection
- Truck detection
- None of the above

Thus, a result adapted to the problem to be solved is obtained.

To achieve this adaptation, it is necessary to make a modification in the part of the code of the neural network in which the images are loaded. Specifically, in the part where the labeling of each image is made. In this way, the original code is modified to meet the needs of this project.

Once the processes to be followed are clear, the case of execution on a single machine has been carried out first. This case is not that it provides novelty or is the objective that this study aims to achieve, but it is necessary to make a first contact with the codes, the executions and the machines used.

For this reason, and without taking into account the modification that is going to be made in the exits, we proceed to the execution of the problem with the original code and having the classification of the ten categories. This is the first step taken in terms of executions to acquire fluency.

Once the different factors surrounding this example have been understood, such as problems arising from communication with the servers or problems in understanding the results, a further step is taken.

The next step has been to carry out the execution of the case in which the outputs have been modified, that is, the classification of the three possible cases that we want to obtain. To make a better compression of the results, a series of ten different launches has been executed in order to analyze the results of each of them.

Performing an average of ten launches, we obtain an execution time of 62 minutes and 52 seconds with a variance of 0.00125, and a precision of accuracy in the test with 95% and a variance of 0.0000004.

Once we have these results, it is time to evaluate these executions but in a distributed environment. This is necessary because in Big Data environments, the execution in a single machine is not usually carried out because the system is restricted to the limitations of said machine, which would be its storage capacity and its processing limitations.

However, it is necessary to emphasize that when performing parallel executions the system can be degraded because the sum of the execution times of all the machines used can be greater than the execution time in a single machine, due to factors of each machine as the exchange of information messages. This will depend on the problem and the data treated.

For distributed execution there is a Hadoop, YARN and Spark environment that has a structure in which there are nodes with different functionalities:

- Client and the Resource Manager, who are in charge of deciding which node will be the driver and which nodes will be the slaves (the decision is random).
- Driver: analyzes, distributes and schedules the work that needs to be done with the data that each specific problem has.
- Compute nodes or executors: they are the nodes that carry out the processing of the data assigned to them. They must report the processing status to the driver.

The results obtained for this execution have been an average accuracy of 86.8% with a variance of 0.000002 and an average training time of 2 hours and 43 minutes, with a variance of 0.0023.

If the two types of executions are compared, it is observed that linear processing has a greater accuracy in the test than the distributed case. However, this last case improves the handling of data, which is very interesting for Big Data issues.

Therefore, a distributed execution is chosen in those cases in which the amount of data to be handled is really huge. Fort he rest of cases, it would not be the best option to choose.

Índice

1. Introducción 16
2. Estado del arte 17
   2.1 Redes Neuronales Artificiales 18
      2.1.1 Arquitectura 19
      2.1.2 Función de activación 19
      2.1.3 Bias 21
      2.1.4 Modos de operación: aprendizaje y ejecución 21
      2.1.5 Redes neuronales convolucionales 22
   2.2 ¿Qué es Deep Learning? 25
   2.3 Motivación 25
3. Planteamiento del problema 27
   3.1 Objetivos 27
4. Planteamiento de la solución 29
   4.1 Comparación de librerías 29
   4.2 Herramientas de procesamiento distribuido 30
   4.3 Elección final 32
5. Experimentos y análisis 34
   5.1 Ejecución en una única máquina 34
   5.2 Ejecución distribuida 35
      4.4.3 Comparativa entre ejecuciones 37
5. Plan de trabajo y presupuesto. 41
   5.1 Diagrama de Gantt 41
   5.2 Presupuesto 42
      5.2.1 Costes de personal 42
      5.2.2 Costes de herramientas 43
      5.2.3 Costes añadidos 44
      5.2.4 Presupuesto total 45
6. Marco regulador 46
   6.1 Programación Python 46
   6.2 Conjunto de imágenes CIFAR-10 46
   6.3 Spark, Tensorflow, Tensorflow on Spark 46
7. Impacto socio-económico 47
8. Conclusiones y trabajos futuros. 48
9. Bibliografía 50

Índice de figuras

Figura 1: Arquitectura de una red neuronal [4]. 19
Figura 2: Función de activación ReLU [5]. 20
Figura 3: Funcionamiento de la función softmax [6]. 21
Figura 4: Algoritmo max-pooling en la capa de reducción [8]. 24
Figura 5: Tiempo de ejecución Hadoop vs Spark [14] 31
Figura 6: Arquitectura ejecución distribuida 36
Figura 7: Frontend Spark 37
Figura 8: Grafo ejecución distribuida 39
Figura 9: Detalle ejecución distribuida 40
Figura 10: Diagrama de Gantt 42
Figura 11: Coste de personal 43
Figura 12: Coste hardware 44
Figura 13: Costes adicionales 44
Figura 14: Coste total del proyecto 45
Figura 15: Funcionamiento Spark Streaming 49

1. Introducción

En la actualidad, se están generando grandes cantidades de datos mediante el uso de los dispositivos electrónicos que nos rodean. Una manera de tratar estos datos es el llamado *Machine Learning* o aprendizaje automático, cuyo objetivo principal es que un sistema sea capaz de aprender y analizar información proporcionada de ejemplo con el objetivo de predecir o generalizar ejemplos futuros sin necesidad de la intervención humana. Este campo está presente en muchas de las aplicaciones que usamos con frecuencia, como los motores de búsqueda de internet, en las redes sociales o en los asistentes de voz virtuales.

Relacionado con esto, se encuentra el *Deep Learning* o aprendizaje profundo. Es una rama de *machine learning* formada por un conjunto de algoritmos y técnicas cuyo propósito es simular el comportamiento que lleva a cabo nuestro cerebro para reconocer imágenes, palabras o sonidos. A lo largo de este trabajo se va a profundizar en este campo para llegar a comprender la importancia que tiene.

Ligado a este concepto, están las redes neuronales artificiales que van a construir nuestra red profunda, ya que emulan el funcionamiento de las neuronas del cerebro.

Una vez que se han definido algunos conceptos básicos, es momento de saber que esta memoria trata, en primer lugar, el estado del arte que rodea a las redes profundas. Después, se plantea el problema a resolver y los objetivos de este estudio. Seguido de las herramientas necesarias, los posibles problemas que puedan surgir y las soluciones propuestas.

Con este esquema, se busca analizar la aplicación de las redes profundas en entornos reales como son las Smart cities, ver el impacto que pueden tener y los problemas que puedan surgir. En concreto, se quieren mejorar los sistemas de seguridad actuales pudiendo llegar a diferenciar entre vehículos permitidos y no permitidos en imágenes de zonas sensibles a ataques.

2. Estado del arte

Una vez comenzaron a desarrollarse las bases de la tecnología actual fue cuando se produjo un fuerte avance en el estudio del cerebro humano. En los años 40 el neurólogo Warren McCulloch y el matemático Walter Pitts propusieron modelos matemáticos y eléctricos de redes neuronales en los que describían cómo funcionaban las neuronas y modelaron una red neuronal simple usando circuitos eléctricos [1]. A finales de la década, Donald Hebb propuso una ley que explicaba a grandes rasgos el aprendizaje neuronal. Esta ley se convirtió en la precursora de las técnicas de entrenamiento de las redes neuronales artificiales actuales.

Durante un periodo de tiempo se perdió el interés sobre el tema ya que una publicación de Minsky y Papert, *Perceptrons: An introduction to Computational Geometry* [2], demostraba que los modelos neuronales artificiales de tipo perceptrón tenían limitaciones prácticas. Por tanto, después de esto muchos fueron los investigadores que decidieron centrarse en los sistemas basados en el conocimiento.

Hubo varios hechos que hicieron resurgir el interés en este campo, pero la clave fue el desarrollo del algoritmo de aprendizaje supervisado para las redes neuronales conocido como *backpropagation,* que ofrecía una potente solución para la construcción de redes más complejas al evitar los problemas observados en el aprendizaje del perceptrón simple [3]. Desde ese momento, este algoritmo ha sido el más usado para el entrenamiento del perceptrón multicapa.

El avance que han experimentado estas redes en los últimos años ha dado como resultado soluciones increíblemente buenas, con rapidez y poca información de partida.

2.1 Redes Neuronales Artificiales

El cerebro, de manera simplificada, puede verse como un conjunto de millones y millones de neuronas interconectadas entre sí mediante sinapsis. Las neuronas artificiales son modelos que tratan de simular el comportamiento de las neuronas biológicas. De esta manera, una red de neuronas realiza un conjunto de operaciones matemáticas sobre un conjunto de datos, produciendo como resultado otro conjunto de datos.

Se puede ver un ejemplo de esta transformación si tomamos una imagen codificada como un conjunto de datos multidimensionales. La red neuronal recibiría a su entrada vectores, por ejemplo, con los píxeles que tenga la imagen. Cabe esperar que a la salida haya tan solo un número indicando si existe la característica buscada o no la hay. Se podría tomar como decisor un valor cercano a 1 que signifique que ha detectado lo que buscamos, un valor cercano a 0 que no lo ha detectado y los valores intermedios como probabilidad.

Los parámetros a diseñar son: cuántas capas ocultas se incluyen, cuántas neuronas poner en cada capa y el algoritmo de entrenamiento. Los dos primeros puntos se suelen decidir a mano y algunas veces mediante prueba y error. Teniendo en cuenta que cuantas más neuronas haya en las capas ocultas o más capas ocultas haya, más compleja será la red y podrá resolver problemas más complejos. Pero a mayor complejidad de red, más costoso será realizar las operaciones matemáticas. En cuanto al algoritmo de entrenamiento a elegir depende del problema, pero no es fácil determinar cuál será el mejor antes de probarlo.

Hay dos tipos de arquitectura de redes neuronales profundas: recurrentes y convolucionales. En este trabajo se van a usar las de segundo tipo ya que las redes neuronales convolucionales están diseñadas para procesar datos estructurados 2D como imágenes o señales de voz presentadas en espectrogramas. Su principal ventaja es que tiene menos parámetros a entrenar que una red multicapa con conexión total del mismo número de capas ocultas, por lo que su entrenamiento resulta más rápido.

2.1.1 Arquitectura

La arquitectura de una red de neuronas se organiza por capas, de manera que hay una capa de entrada, una capa de salida y capas internas ocultas que pueden variar de número. La capa de entrada recibe un conjunto de datos multidimensionales, la capa de salida muestra el resultado una vez la red ha hecho las operaciones matemáticas y las capas ocultas se encargan de realizar dichas operaciones.

La conexión entre las capas tiene asociado un número que se llama peso. Así, se multiplican los valores de una neurona por los pesos de sus conexiones salientes. Por tanto, cada neurona de la siguiente capa recibe números de varias conexiones entrantes y los suma todos. Los pesos de cada neurona se inicializan generalmente de forma aleatoria. En la siguiente imagen podemos verlo gráficamente.

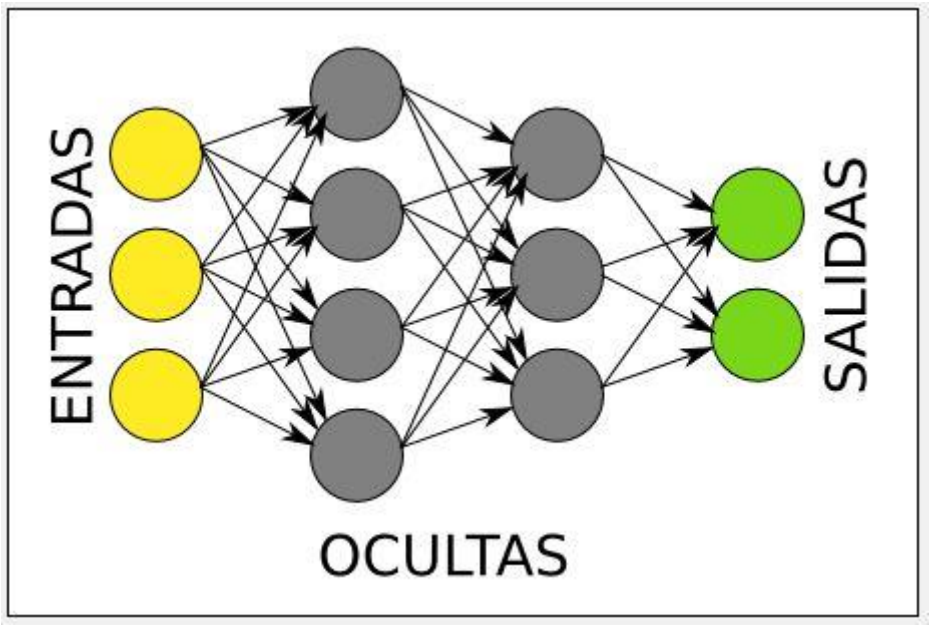


Figura 1: Arquitectura de una red neuronal [4].

2.1.2 Función de activación

La función de activación es la encargada de decidir si una neurona debe activarse o no, para ello realiza los cálculos pertinentes en un nodo para hallar la salida en función de una entrada. Esto se aplica a todas las capas de la red neuronal excepto a la capa de entrada y consta de tres pasos: la suma de los valores que llegan al nodo de las

conexiones entrantes, dicho valor lo transforma mediante una fórmula y, por último, produce un nuevo número. Se puede observar mejor en la siguiente fórmula:

$$Y = Activation\ (\Sigma(weight * input) + bias)$$

Las funciones de activación más comúnmente usadas son: la función lineal (la salida se fija entre -1 y 1), la función sigmoidea (la salida se define en un rango entre 0 y 1) y la función tangente hiperbólica (la salida está comprendida en un rango entre -1 y 1)[1].

En el presente trabajo se aplica una función de activación diferente, la función lineal rectificada o ReLU, pues se ha demostrado que para algunos casos se obtienen unos mejores resultados [5]. A continuación, se muestra la gráfica que describe el comportamiento de ReLU. En nuestro caso, z será igual a 1.

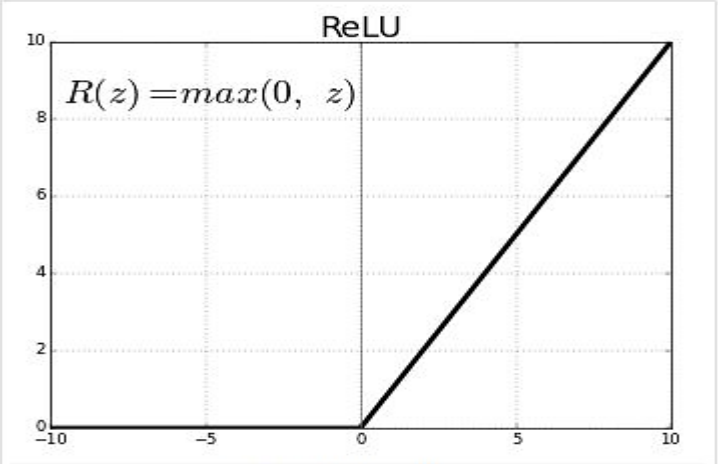


Figura 2: Función de activación ReLU [5].

En la última capa de la red neuronal se aplica la regresión softmax, o regresión lineal multinomial, cuyo objetivo consiste en conseguir la probabilidad de las salidas, es decir, la probabilidad de que el elemento analizado sea o no el que buscamos. Esta función adapta las salidas para que el valor esté entre 0 y 1. Y divide cada salida de modo que la suma total de las salidas sea igual a 1. La probabilidad es cercana a 0 cuando el objeto no es el que buscamos y es cercana a 1 cuando hay muchas probabilidades de que sí lo sea.

Figura 3: Funcionamiento de la función softmax [6].

2.1.3 Bias

El *bias* (sesgo), ayuda a desplazar la función de activación hacia la izquierda o hacia la derecha para compensar el efecto que tiene el peso de la neurona en el cálculo a realizar.

Normalmente el valor del *bias* se fija a 1 o -1 y lo que varía es el peso de la neurona cuyo valor es determinado por el método del descenso de gradiente. El objetivo de este método es minimizar las funciones y para ello comienza con un conjunto inicial de valores y modifica iterativamente dichos valores de manera que minimice la función.

2.1.4 Modos de operación: aprendizaje y ejecución

El funcionamiento de una red neuronal se divide en dos: una fase de aprendizaje y una fase de ejecución [7]. Con la fase de aprendizaje la red es entrenada de una forma determinada y una vez alcanzado un nivel de entrenamiento adecuado, se procede a la fase de ejecución, donde la red lleva a cabo las tareas para las que ha sido preparada.

En la fase de aprendizaje se parte de unos pesos sinápticos aleatorios y mediante iteraciones se busca un conjunto de pesos que permitan a la red desarrollar adecuadamente la tarea para la que ha sido creada. Principalmente, hay tres formas de llevar a cabo dicho aprendizaje:

- **Aprendizaje supervisado**. Se presenta a la red un conjunto de patrones de entrada y su salida esperada. Los pesos se van modificando de forma proporcional al error que se produce entre la salida real y la esperada. En

este tipo de aprendizaje encontramos las técnicas de clasificación y las de regresión. La primera determina, para un elemento, en qué categoría se encuentra. La segunda, predice para un elemento, cuál es su valor aproximado.

- **Aprendizaje no supervisado**. Se presenta a la red un conjunto de patrones de entrada. No se proporciona información de la salida esperada. En este caso se ajustan los pesos en base a la correlación existente entre los datos de entrada. Se encuentran en este tipo, las técnicas de agrupamiento o *clustering* y los conjuntos frecuentes. La primera técnica establece en un conjunto de elementos, cuáles son semejantes entre sí y la segunda, determina en una lista de sucesos qué elementos se repiten con frecuencia.
- **Aprendizaje por refuerzo**. Este tipo es una combinación de las dos anteriores. Se le presenta a la red un conjunto de patrones de entrada y se le indica a la red si la salida obtenida es o no correcta. Pero no se le proporciona el valor de la salida esperada. Esta forma de aprendizaje es útil para casos en los que se desconoce la salida exacta que debe proporcionar la red.

En la fase de evaluación se evalúan nuevos datos que la máquina no tiene reconocidos, llamado conjunto de datos de test. De esta evaluación se espera que la máquina proporcione una solución válida en base a lo aprendido en la fase de aprendizaje. Es en esta fase del proceso donde se verifica que la máquina entrenada no ha sufrido sobreajuste y generaliza lo suficientemente bien como para poder clasificar muestras desconocidas con un error aceptable. En algunos casos, las máquinas pueden ser modificadas en base a los errores cometidos en la fase de producción, denominadas máquinas adaptativas. Pero ese no es el caso que sigue este estudio.

2.1.5 Redes neuronales convolucionales

Este tipo de redes son similares a las redes neuronales ordinarias, tienen pesos, *bias*, reciben una entrada con la que realiza un producto escalar y sobre la que aplica una función de activación. Se usan principalmente para resolver el tratamiento de imágenes.

Aunque las redes neuronales con algoritmos y topologías estándar son capaces de manejar imágenes, cuando aumentan de tamaño y calidad se vuelven intratables. Al tratarse de neuronas totalmente conectadas, provocaríamos un desperdicio de recursos, así como un rápido sobreajuste.

Las redes neuronales convolucionales trabajan dividiendo y modelando la información en partes más pequeñas, y combinando esta información en las capas más profundas de la red. Dicho de otro modo, trabajan modelando de forma consecutiva pequeñas piezas de información y luego combinando esta información en las capas más profundas de la red. Una manera de entenderlas es que la primera capa intentará detectar los bordes y establecer patrones de detección de bordes. Luego, las capas posteriores tratarán de combinarlos en formas más simples y, finalmente, en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc. Las capas finales intentarán hacer coincidir una imagen de entrada con todos los patrones y llegar a una predicción final como una suma ponderada de todos ellos. Así, se pueden modelar complejas variaciones y comportamientos dando predicciones bastante precisas.

Están formadas por tres capas diferenciadas: la capa convolucional, la capa de reducción o *pooling* y la capa clasificadora [8].

- **Capa convolucional.** La diferencia de este tipo de redes es la llamada operación "convolución" en algunas de sus capas; en lugar de utilizar la multiplicación de matrices que se aplica generalmente. La operación de convolución recibe como entrada la imagen y luego aplica sobre ella un filtro que nos devuelve un mapa de características de la imagen original, así reducimos el tamaño de los parámetros. La convolución aprovecha tres ideas importantes que ayudan a mejorar cualquier sistema: interacciones dispersas (al aplicar el filtro de menor tamaño sobre la entrada original, se reducirá la cantidad de parámetros y cálculos), los parámetros compartidos (se comparten los parámetros entre los distintos tipos de filtros, ayudando a mejorar la eficiencia del sistema) y las representaciones equivariantes (indican que si las entradas cambian, las salidas van a cambiar de forma similar).
- **Capa de reducción.** Generalmente se sitúa después de la capa convolucional. Su principal uso radica en la reducción de las dimensiones espaciales (ancho x alto)

del volumen de entrada para la siguiente capa convolucional. No afecta a la dimensión de profundidad. Esta operación también se llama reducción del muestreo, ya que la reducción de tamaño conduce también a una pérdida de información. Sin embargo, una pérdida de este tipo puede ser beneficioso para la red por dos razones: la disminución en el tamaño conduce a una menos sobrecarga del cálculo para las próximas capas de la red; también ayuda a reducir el sobreajuste. La operación que se suele realizar en esta capa es max-pooling, que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se va quedando con el máximo valor.

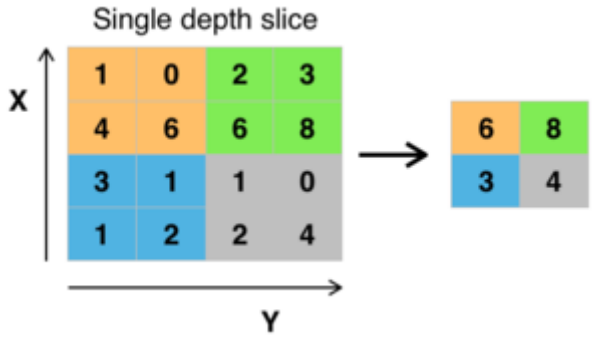


Figura 4: Algoritmo max-pooling en la capa de reducción [8].

- **Capa clasificadora.** Última capa totalmente conectada en la que cada píxel se considera como una neurona separada al igual que en una red neuronal ordinaria. Esta capa tendrá tantas neuronas como número de clases que se debe predecir.

Algunas de las aplicaciones más conocidas son: reconocimiento de números y letras manuscritos incluyendo caracteres arábigos y chinos; detección y borrado automático de rostros y placas vehiculares para protección de la privacidad; identificación de expresiones faciales; evasión de obstáculos para robots móviles; restauración de imágenes o en combinación con algoritmos evolutivos.

2.2 ¿Qué es Deep Learning?

Continuando la descripción sobre *Deep Learning* y su propósito de que un sistema pueda replicar el comportamiento de las neuronas, es momento de explicar: cómo surge este término y cómo funciona internamente.

Aunque el fenómeno en sí surge en la década de los 50 cuando Arthur Samuel desarrolla el primer programa que permite aprender a un ordenador, no se surge el término tal y como se usa en la actualidad hasta 2006 cuando Geoffrey Hinton lo emplea para explicar la mejor capacidad de aprendizaje con la que cuentan las redes neuronales profundas [9].

Los algoritmos funcionan en base a las técnicas de redes neuronales y estas redes están formadas por capas en las que, en cada capa se extrae y se procesan información útil para la siguiente capa, de forma que es capaz de obtener como resultado un aprendizaje minucioso de la información de entrada y poder extraer conclusiones relevantes. En cada capa el algoritmo aprende una nueva característica de la entrada.

Existen numerosas aplicaciones donde *Deep Learning* puede agilizar los procesos que antes se realizaban por el ser humano como pueden ser las traducciones automáticas, clasificar objetos en fotografías, añadir sonido a películas mudas, colorear imágenes en blanco y negro, entre otras. Algunos otros ejemplos se conseguirán a medida que este campo avance en el tiempo como traducir discursos en tiempo real o focalizar objetos en movimiento en fotografías.

2.3 Motivación

El impacto generado por los fenómenos *Big Data*, *Machine Learning*, *Deep Learning*, Internet de las cosas, entre otros, ha sido espectacular. Ha revolucionado la tecnología y mucho más aún. Obtenemos datos de todo lo que nos rodea y, con ello, las máquinas pueden elegir nuestras preferencias como canciones que nos gustarán, la temperatura adecuada a nuestro cuerpo o la ruta que seguiremos en el próximo destino.

Big Data surgió de la generación y recolección de datos producidos en forma masiva por usuarios y dispositivos. Las imágenes forman parte de estos datos y es necesario que sean tratadas de forma adecuada para obtener resultados útiles.

Estas son las razones que han inspirado la realización de un estudio como este, para saber cómo se originaron estos conocimientos tecnológicos, cómo se comenzaron a aplicar a los dispositivos, emular un ejemplo y llegar a comprender cómo funciona internamente. Y, por supuesto, saber qué nos espera más adelante. A medida que pase el tiempo, se irá avanzando en estos temas y descubriremos qué más puede hacer la tecnología por nosotros.

Las empresas incorporan cada vez más este fenómeno para orientar su marketing a los gustos de la población y poder conseguir así más clientes o resultar más atractivos. Empresas como Google dicen "*No tenemos departamento que no se beneficie del Machine Learning*". Facebook, por su parte, usa *Deep Learning* para orientar los anuncios en función de los gustos de los usuarios e identificar rostros y objetos en fotos y vídeos. Y Microsoft lo emplea en proyectos de reconocimiento de voz.

Conocer los grandes avances que se están produciendo, saber qué hay detrás de los procedimientos que se llevan a cabo, los distintos programas que se usan y conocer qué pasará más adelante, es la motivación que ha inspirado este trabajo, la inquietud por saber.

3. Planteamiento del problema

El problema de los datos masivos se centra principalmente en cómo tratar la gran cantidad de datos para obtener buenos resultados. Esto es, los datos no son siempre buenos, no están siempre completos y no siempre es mejor tener mayor cantidad de datos para obtener mejores resultados. Esto último es un concepto erróneo que suele pensarse, pero no es así.

Una vez tenemos los datos en crudo y antes de proceder a analizarlos, hay que definir el objetivo del estudio y si la tarea será clasificar o predecir, por ejemplo. Hecho este paso, se procede a comprender los datos de los que se dispone, para ello puede ser útil realizar visualizaciones. Después se realiza un pre-procesado, en el que se realizan tareas como transformación de variables, generación de atributos y selección, entre otras.

Una vez los datos están a nuestro gusto, se procede al modelado. Este es el momento de elegir la técnica de aprendizaje para entrenar el modelo. A continuación, es necesario evaluar los resultados obtenidos para estimar el rendimiento y el error del modelo y, de esta manera, proceder a la elección del que ofrezca mejores resultados. Finalmente, se procederá a la integración en un sistema informático. Todas estas etapas son las que forman lo que se llama minería de datos.

3.1 Objetivos

El objetivo principal de este estudio se centra en el uso las tecnologías actuales de redes profundas y los grandes avances conseguidos para incrementar la seguridad del ciudadano y, así, ser una ayuda añadida a los cuerpos de seguridad.

Para ello, la idea es crear un sistema que genere una alarma en un sistema de vigilancia pasiva en los puntos clave de las ciudades donde es prioritario tener un control de seguridad. La alarma se produciría ante la entrada de un vehículo de gran tamaño en zonas de gran aglomeración para estar alerta de lo que pudiera ocurrir, evitar altercados y ser eficientes en el momento de actuar. Para ello, se aplicará la clasificación de las

imágenes recogidas por las cámaras que actualmente tienen monitorizados los barrios más transitados de la ciudad.

Se pueden considerar objetivos adicionales ya que este estudio implica, además: conocer cómo trabaja *Spark* con los datos que se le proporcionan; y el conocimiento que el estudiante adquiere con la realización de este proyecto.

Una vez que están claros los objetivos y las herramientas que son necesarias, es momento de detallar qué pasos hay que seguir para obtener el resultado buscado. Para ello, se define a continuación el desarrollo que ha tenido el estudio.

4. Planteamiento de la solución

Una vez se han fijado los objetivos del estudio, se procede a plantear una solución adecuada que corresponda con lo detallado hasta ahora.

En primer lugar, es necesario el uso de diversas herramientas para construir esta implementación del sistema de vigilancia pasiva. Se deben recoger las imágenes de las cámaras de aquellas zonas de la ciudad susceptibles de ataques de cualquier tipo con vehículos de gran tamaño. De esta manera, el sistema generaría una alarma al procesar una imagen en la que aparece un vehículo considerado como no permitido, para notificar a un responsable del cuerpo de seguridad y ellos actuar, en caso de ser necesario, con rapidez y eficacia.

Sabemos que nuestro objetivo es clasificar imágenes para llegar a generar la alarma o no generarla. Por tanto, como ya hemos comentado, se usarán las redes neuronales convolucionales. Pero como se van a manejar grandes cantidades de datos, es necesario aplicar las herramientas de *Big Data*.

4.1 Comparación de librerías

Para desarrollar la arquitectura y proceder al entrenamiento de aprendizaje profundo se han comparado tres librerías muy comúnmente usadas en estos entornos. En la comparación se han valorado sus ventajas y desventajas de la siguiente manera:

- **Caffe**: es una de las primeras librerías que se comenzaron a utilizar. Tiene un buen rendimiento, se centra en redes neuronales convolucionales y la posibilidad de implementar CPU y GPU. El código fuente es fácil de interpretar y permite la visualización de la red. Pero tiene escasa documentación y no es recomendada para grandes redes o nuevas arquitecturas [10].
- **Theano**: también es una de las librerías más veteranas. Se considera una herramienta más manual y flexible pero más lenta que otras. No soporta GPU originalmente aunque sí mediante el uso de librerías externas y no hay muchos

modelos pre-entrenados. Estos hechos están haciendo que actualmente, no sea muy utilizada [11].

- **Tensorflow**: es una librería desarrollada por Google que cuenta con muchos desarrolladores y usuarios. Soporta multi-GPU, es flexible y realiza una compilación de modelos más rápida que las opciones basadas en *Theano*. Sin embargo, inicialmente es más lento en algunas funciones, pero está mejorando [12].

En base a estas características y en base a los requisitos que son necesarios evaluar en nuestro sistema que son: el tiempo de respuesta y el nivel de fiabilidad que proporcionen, se ha decidido elegir la librería *Tensorflow*.

4.2 Herramientas de procesamiento distribuido

En el problema planteado, es indispensable contar con una solución *big data*, haciendo uso de un *cluster* de servidores interconectados entre sí con un software de gestión para el reparto de datos y tareas. Las plataformas más comunes para llevar a cabo esta solución son *Hadoop* y *Spark*. Se realiza a continuación una comparativa para la elección de una de ellas.

- **Hadoop:** plataforma para implementar soluciones *Big Data* aplicando el algoritmo de *Map-Reduce*. Cuenta con software abierto, escalable y tolerante a errores. Procesa de forma eficiente distintos volúmenes de datos en un *cluster* de hardware básico. Su velocidad de procesamiento es más lenta de lo deseable ya que lee y escribe en disco. En cuanto a la administración, se dispone de diferentes motores, por lo que es tedioso administrar muchos componentes. El funcionamiento en tiempo real reporta fallos, ya que se diseñó para realizar el procesamiento por lotes en cantidades voluminosas de datos. Pero sí cuenta con tolerancia a fallos, sin necesidad de reiniciar la aplicación en caso de fallo de algún componente. Su gran importancia es que fue de las primeras plataformas que hacían posible el tratamiento masivo de datos y se convirtió en la base de referencia para

las plataformas de *Big Data*. Además, se puede destacar que cuenta con un buen nivel de seguridad [13].

- **Spark**: es una plataforma para implementar aplicaciones de *Big Data*, no solo basadas en el algoritmo de *Map-Reduce*, sino también para aprendizaje máquina o clasificación de control de flujos de datos, por ejemplo. Ejecuta aplicaciones hasta cien veces más rápidas en memoria y hasta diez veces más rápido en disco, en comparación con *Hadoop*. Esto se debe a la reducción de ciclos de lectura/escritura en el disco y al almacenamiento de datos en memoria. También resulta fácil de programar ya que tiene multitud de operadores de alto nivel. Es un motor de análisis de datos completo, por lo que no es necesario administrar diferentes componentes para cada necesidad. Cuenta con la posibilidad de procesar datos en tiempo real. También tiene tolerancia a errores, no necesita reiniciar la aplicación desde cero en caso de error. *Spark* cuenta con menor seguridad porque admite una única autenticación mediante la contraseña secreta compartida [14].

Spark es la herramienta que se va a implementar debido a su mayor flexibilidad y velocidad. A continuación, se puede ver la diferencia de velocidad en cuanto a procesamiento entre ambas herramientas. Observando una diferencia muy significativa.

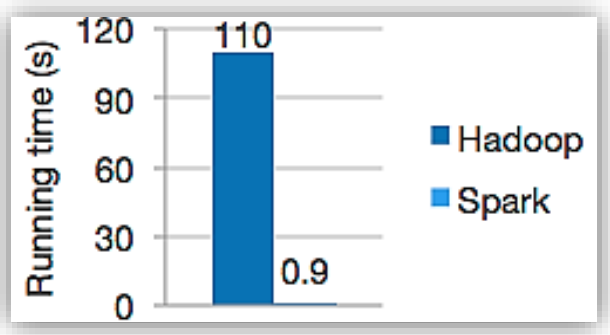


Figura 5: Tiempo de ejecución Hadoop vs Spark [14]

4.3 Elección final

Es momento de unir las dos herramientas anteriormente escogidas para la implementación de la solución. Existen las siguientes herramientas: *Tensorflow on Spark, Theano on Spark y Caffe on Spark*. Que son librerías que se implementan sobre *Tensorflow, Theano* y *Caffe*, respectivamente, pero que permiten coger los datos y desplegarlos en un entorno *Big Data*. Con ellos, se entrena la máquina profunda. En base a la elección que ya se ha efectuado, se escoge *Tensorflow on Spark*, que ofrece aprendizaje profundo escalable para los *clusters*. Permite el entrenamiento distribuido de *Tensorflow* y la inferencia en *clusters* de *Spark*. Busca minimizar la cantidad de cambios de código necesarios para ejecutar los programas existentes. Proporciona beneficios importantes como se puede leer en la página web del desarrollador [15]:

- Migra fácilmente todos los programas existentes en *Tensorflow* con menos de diez líneas de cambio de código.
- Admite todas las funcionalidades de *Tensorflow*: entrenamiento síncrono/asíncrono, paralelismo modelo/datos, inferencia y *Tensorboard*.
- La comunicación directa de servidor a servidor consigue un aprendizaje más rápido.
- Implementado fácilmente en la nube o en instalaciones CPU y GPU.

El sistema desarrollado se ha hecho de forma transparente para poder implementarlo en cualquier *cluster*, como podría ser AWS (*Amazon Web Services*). Sin embargo, el entorno de simulación ha sido el clúster del departamento de Teoría de la Señal y la Comunicación. Para ello fue necesario pedirle al departamento técnico un usuario personalizado para poder acceder a dichos servidores. A través de esta cuenta y mediante programas de acceso remoto, se accedían a ellos para lanzar las ejecuciones necesarias.

En primer lugar, se ha probado la eficacia de *Tensorflow* con un conjunto de datos etiquetados, CIFAR-10 [16] [17]. Dichos datos son imágenes y cuenta con 60000, siendo 50000 de ellas para entrenar la red y el resto, 10000, para la evaluación del modelo. Es importante no usar los mismos datos para ambas fases, ya que de esta manera no estaríamos proporcionando datos nuevos a la máquina y la evaluación

realizada sería falsa. En base a esta arquitectura, el modelo tiene diez categorías de clasificación, pero en este caso se procede a modificar la etapa de salida para que decida 3 categorías: coche, camión o ninguna de las anteriores.

Para modificar la etapa de salida se han hecho pequeñas modificaciones en el código de manera que solo considere las etiquetas correspondientes a coche o camión y, el resto que no correspondan a estos dos objetos, se agruparán en una tercera que será “no coincide”.

5. Experimentos y análisis

5.1 Ejecución en una única máquina

En este apartado se procede a explicar los distintos pasos seguidos en la ejecución del sistema en una sola máquina. Aunque esto no aporta ninguna novedad en cuanto al problema que se plantea en este estudio, es necesario establecer una base de referencia con los sistemas convencionales y saber así, cómo de buenos resultan en cuanto a velocidad y precisión, y poder realizar una comparación con los experimentos de procesamiento distribuido. Se llevan a cabo experimentos en dos fases:

- **Ejecución sin realizar modificaciones**. De esta manera se puede valorar cómo actúa la ejecución, los tiempos necesarios para llevarlo a cabo y, lo principal, para adquirir soltura y manejo sobre el programa.
- **Ejecución modificando las salidas**. Una vez conocido el sistema es necesario adaptarlo al problema concreto que se quiere tratar. Para ello no se requiere obtener a la salida las clasificaciones de los diez tipos de imágenes disponibles, sino solo de las tres que resultan interesantes: detectar coche, camión o ninguna de las anteriores. Esto se consigue con una pequeña modificación en el momento del etiquetado de las imágenes. En lugar de etiquetar cada una de las imágenes con sus etiquetas iniciales que las asociaban a un grupo exacto de objetos, se realiza la asociación solo para etiquetar coches y camiones como se realizaba anteriormente y, el resto de elementos se unen en único grupo en el que ya no se hace distinción entre esos objetos.

  Para hacer una mejor evaluación del funcionamiento del sistema se realiza un juego de diez lanzamientos distintos para calcular una media de tiempos y de precisión del clasificador.

Hay que destacar que la máquina utilizada es un servidor dedicado con 64 Gb de memoria y 16 núcleos para ejecución.

El problema de la ejecución en una sola máquina es que en entornos *Big Data*, las ejecuciones tienen las limitaciones de procesamiento de esa única máquina y su

capacidad de almacenamiento. Por este motivo se suele diseñar la implementación de la ejecución y la distribución de datos en múltiples equipos.

Sin embargo, cuando se realizan ejecuciones en paralelo el rendimiento puede degradarse, ya que la suma de los tiempos de ejecución de todas las máquinas que realizan el procesamiento distribuido puede ser mayor que en el caso de una sola máquina, debido a factores que son necesarios en cada máquina como el intercambio de información y la configuración inicial de las máquinas remotas. Esto puede justificarse mediante la Ley de Amdahl [18], que evalúa como cambia el rendimiento al variar la forma de ejecución.

5.2 Ejecución distribuida

Los resultados obtenidos están bastante lejos de poder ayudar eficientemente a detectar vehículos peligrosos con rapidez. Este hecho es el que hace necesario el procesamiento distribuido de los datos mediante *Spark*.

Se estructura de manera que una máquina es la encargada de analizar, distribuir y programar el trabajo a realizar con los datos, llamada *driver*, y el resto de máquinas empleadas son los nodos de cómputo, o *executors*, éstos son los que llevan a cabo el procesamiento de los datos parciales que le son asignados y, en función de ello, deben informar del estado de cómputo al *driver*. Adicionalmente, contamos con el *Resource Manager* y el Cliente, nodos necesarios para controlar las máquinas físicas y elegir cuál de ellas realiza el papel de *driver* y cuáles se convierten en nodos de cómputo. Es importante destacar que se va a trabajar con una GPU por máquina y el entorno de trabajo es *Hadoop, Yarn* y *Spark*. El último escalón que tendrá esta arquitectura será HDFS, que se usa para el intercambio de información. En la siguiente imagen se puede ver un esquema de la arquitectura:

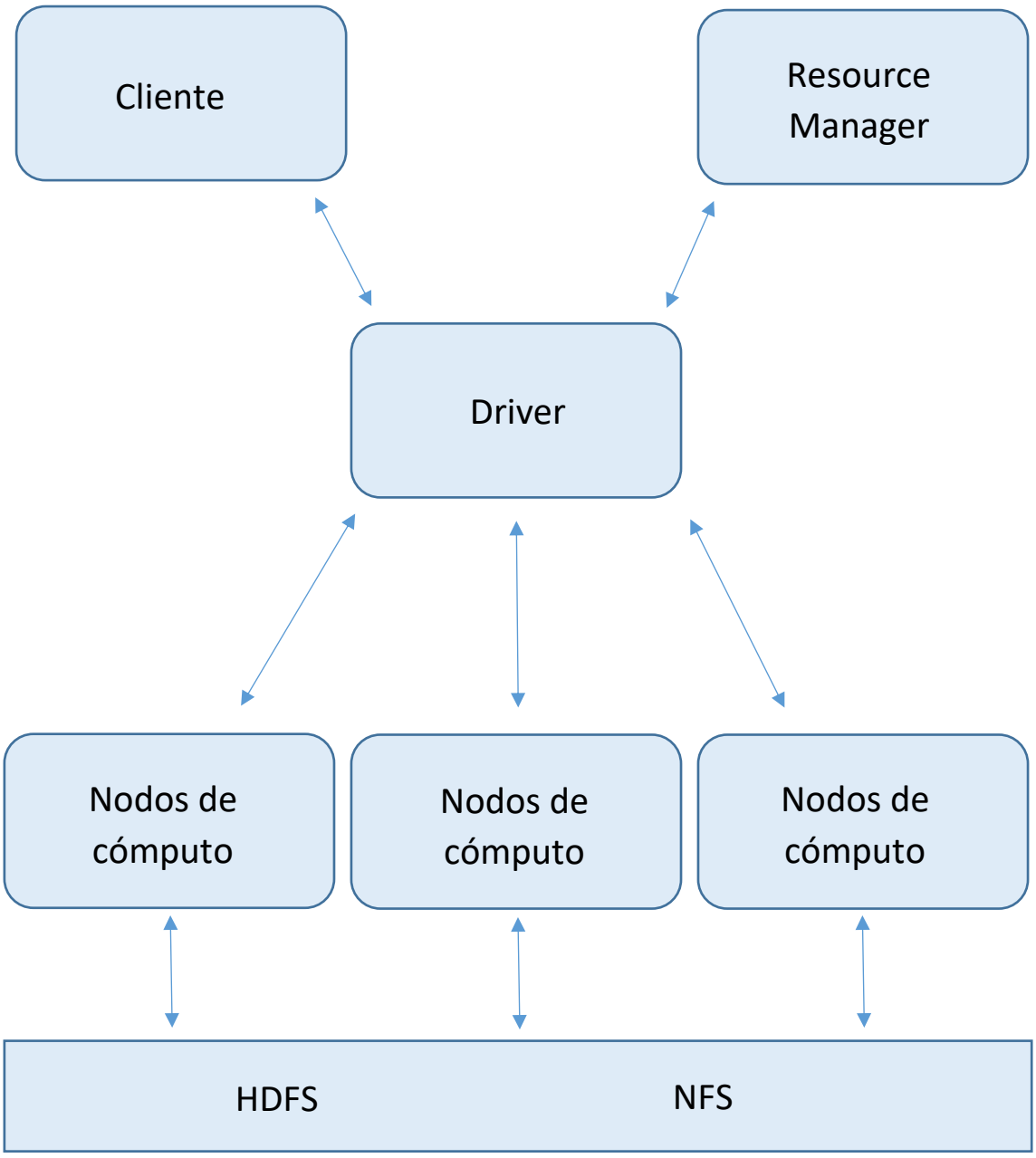


Figura 6: Arquitectura ejecución distribuida

Para realizar una mejor evaluación se lanzan diez ejecuciones, una sobre cada entrenamiento realizado anteriormente. De aquí nos interesa conocer la precisión de la evaluación realizada y cuánto tiempo ha llevado el entrenamiento.

La configuración del entorno necesaria para lanzar dichas ejecuciones se basa en disponer de la base de datos CIFAR-10, la cual está descargada en un sistema de

almacenamiento compartido, y realizar el etiquetado con la librería de *Tensorflow on Spark* modificada para el caso de las tres salidas objetivo de este sistema. Para que dicha ejecución se lleve a cabo, se debe indicar en un comando en el terminal los datos necesarios: cuál es el frontend de Spark, los recursos que se necesitan de los servidores y los parámetros propios de ejecución.

El progreso que siguen las máquinas en *Spark* puede seguirse mediente un *frontend* del que se dispone y que está desplegado en el departamento de la universidad. A través de él puede verse cómo *Spark* distribuye los datos en los tres *executors* asignados. En la siguiente figura puede verse el detalle de los equipos implicados:

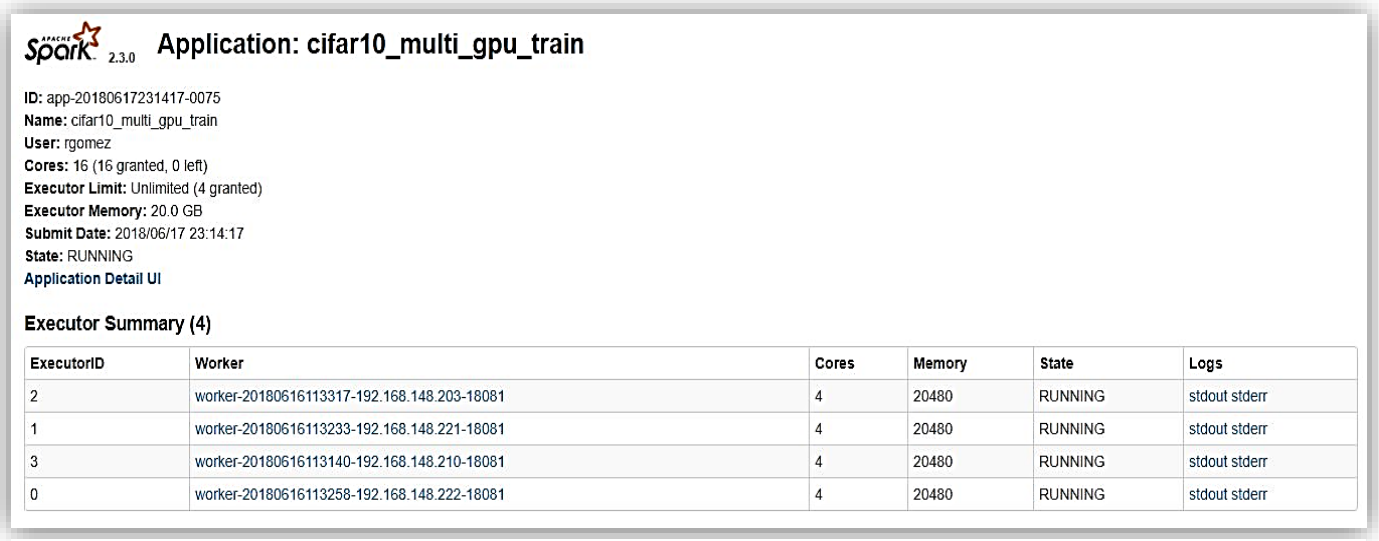


Figura 7: Frontend Spark

4.4.3 Comparativa entre ejecuciones

La finalidad de realizar una comparativa entre los dos tipos de ejecución lanzados, es decidir cuál de ellos se adapta mejor al problema en cuestión.

Se puede intuir que el caso de ejecución distribuida mejoraría las condiciones que presenta la ejecución en una máquina ya que, como se ha comentado anteriormente, los datos se distribuyen para realizar un mejor procesamiento y no quedar limitado por las funcionalidades de una sola máquina.

Sin embargo, también se ha visto anteriormente que el caso distribuido añade tiempo extra en comparación con el caso lineal ya que se requiere un intercambio de información con cada una de las máquinas.

Los resultados obtenidos en la ejecución lineal son un tiempo medio de entrenamiento de 62 minutos y 52 segundos con una varianza de 0,00125, y una precisión en la clasificación de 95% con una varianza de 0,0000004. Mientras que en las ejecuciones distribuidas, se obtiene una precisión media en la clasificación del 86,8% con una varianza de 0,000002 y un tiempo medio de entrenamiento de 2 horas y 43 minutos, con una varianza de 0,0023. Destacar que en los resultados se obtiene una varianza ya que estamos ante un problema no determinista.

Según los datos obtenidos, se observa que el caso lineal requiere menos tiempo para realizar un entrenamiento de los datos que el caso paralelizable. También se obtiene que el caso lineal consigue una mejor precisión en la evaluación que el caso distribuido.

Estos dos hechos hacen favorable el uso del caso lineal, por lo que el uso de *Spark* no resulta necesario para el caso en concreto que se está tratando. Esto se debe a que los datos que se manejan son muchos pero no lo suficientemente grandes como para que usar *Spark* sea ventajoso.

Es conveniente comentar que *Spark* no realiza balanceo de carga, por lo que puede asignar más carga a unas máquinas que a otras. Y, puesto que el tiempo que tarda la ejecución en terminar es el tiempo que tarda la máquina que más se demora, si no se realiza un buen balanceo de datos, puede no resultar una buena técnica. Este hecho ha influido sobre los resultados obtenidos en cuanto al tiempo medio calculado. Dicho problema se debe principalmente a la implementación de la granja, aunque puede verse también afectado por el algoritmo empleado.

En las siguientes figuras se puede observar el grafo que se genera durante la ejecución en el que se ven las máquinas implicadas y su tiempo de cómputo, y datos adicionales de la ejecución.

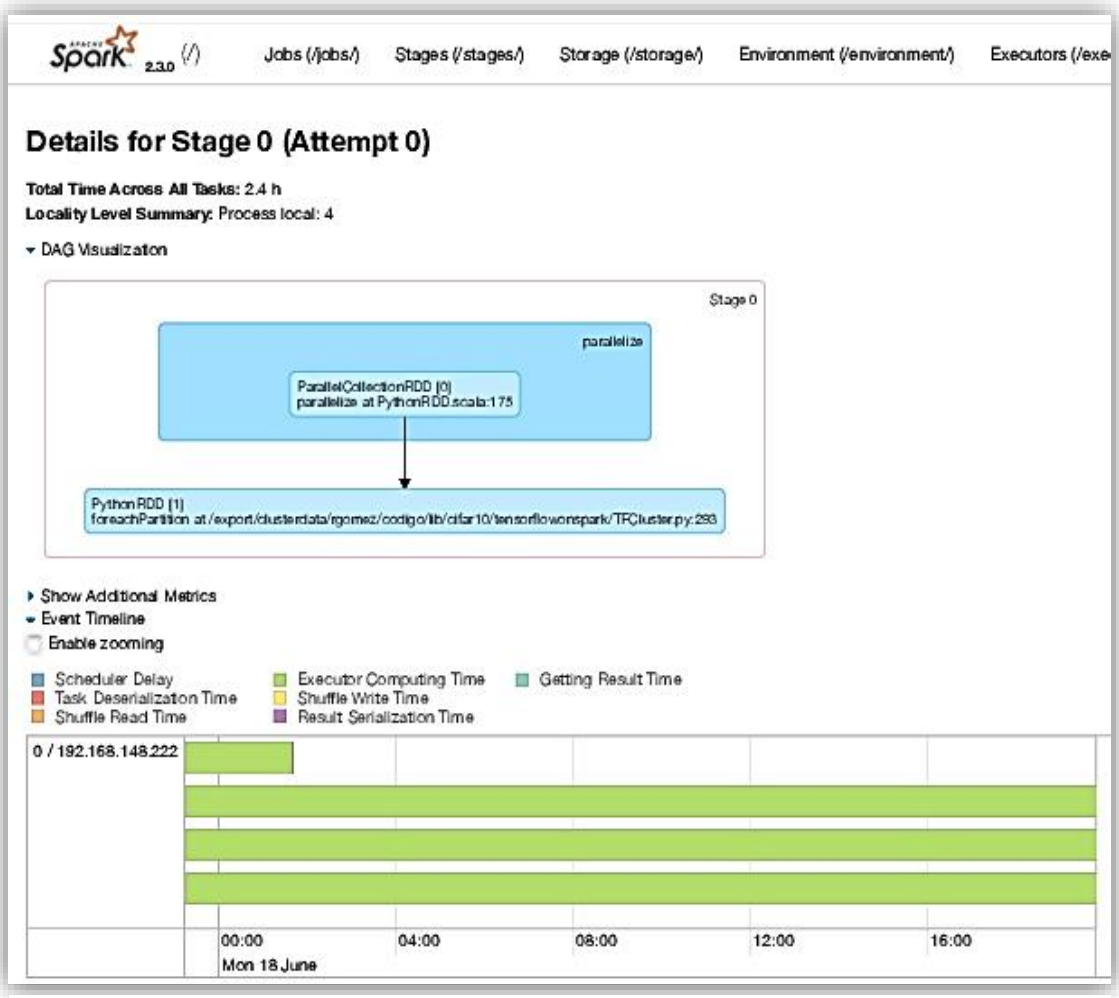
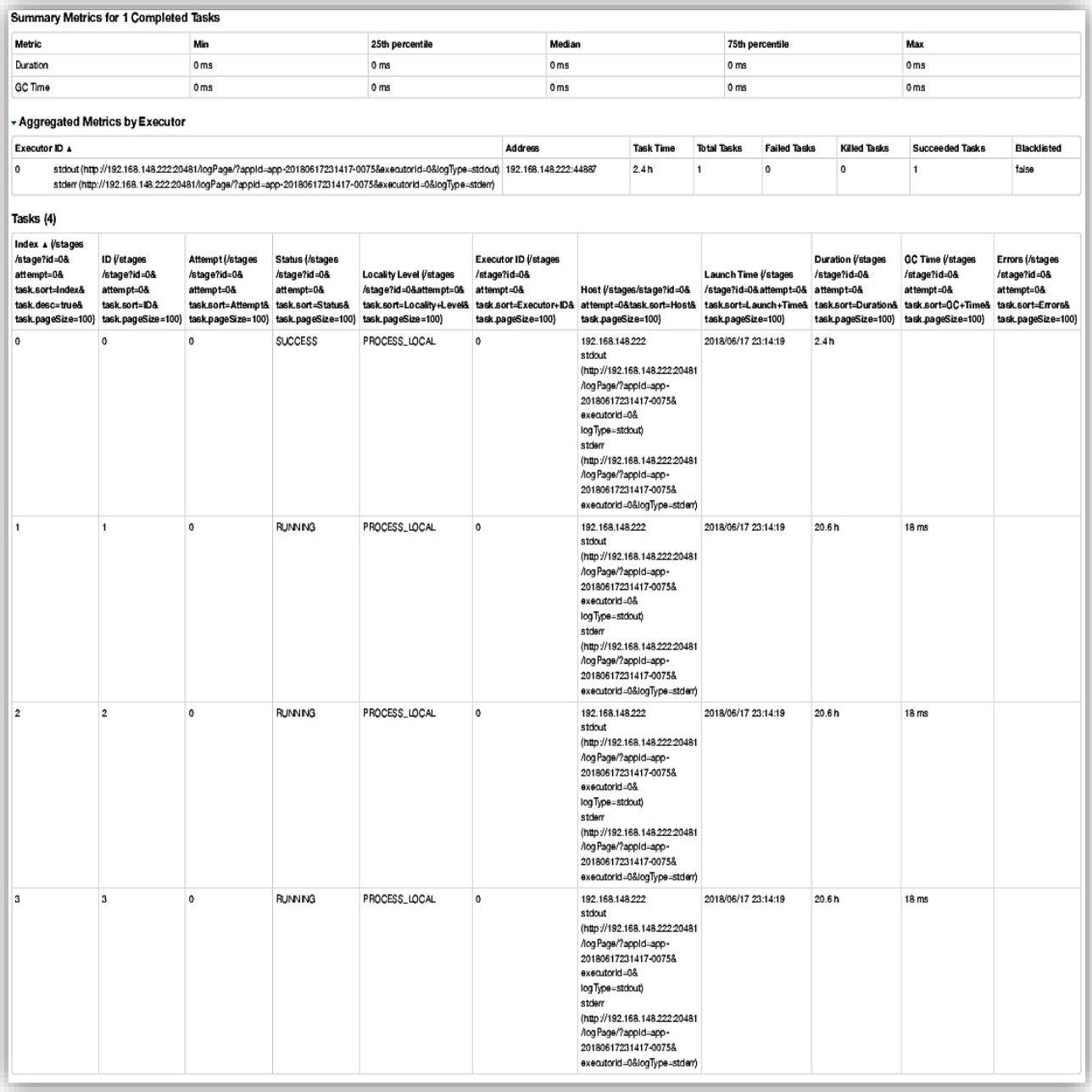


Figura 8: Grafo ejecución distribuida



Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	0 ms	0 ms	0 ms	0 ms	0 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Blacklisted
0 stdout (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stdout) stderr (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stderr)	192.168.148.222:44887	2.4 h	1	0	0	1	false

Tasks (4)

Index ▲ (/stages/stage?id=0&attempt=0&task.sort=Index&task.desc=true&task.pageSize=100)	ID (/stages/stage?id=0&attempt=0&task.sort=ID&task.pageSize=100)	Attempt (/stages/stage?id=0&attempt=0&task.sort=Attempt&task.pageSize=100)	Status (/stages/stage?id=0&attempt=0&task.sort=Status&task.pageSize=100)	Locality Level (/stages/stage?id=0&attempt=0&task.sort=Locality+Level&task.pageSize=100)	Executor ID (/stages/stage?id=0&attempt=0&task.sort=Executor+ID&task.pageSize=100)	Host (/stages/stage?id=0&attempt=0&task.sort=Host&task.pageSize=100)	Launch Time (/stages/stage?id=0&attempt=0&task.sort=Launch+Time&task.pageSize=100)	Duration (/stages/stage?id=0&attempt=0&task.sort=Duration&task.pageSize=100)	GC Time (/stages/stage?id=0&attempt=0&task.sort=GC+Time&task.pageSize=100)	Errors (/stages/stage?id=0&attempt=0&task.sort=Errors&task.pageSize=100)
0	0	0	SUCCESS	PROCESS_LOCAL	0	192.168.148.222 stdout (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stdout) stderr (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stderr)	2018/06/17 23:14:19	2.4 h		
1	1	0	RUNNING	PROCESS_LOCAL	0	192.168.148.222 stdout (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stdout) stderr (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stderr)	2018/06/17 23:14:19	20.6 h	18 ms	
2	2	0	RUNNING	PROCESS_LOCAL	0	192.168.148.222 stdout (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stdout) stderr (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stderr)	2018/06/17 23:14:19	20.6 h	18 ms	
3	3	0	RUNNING	PROCESS_LOCAL	0	192.168.148.222 stdout (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stdout) stderr (http://192.168.148.222:20481/logPage/?appId=app-20180617231417-0075&executorId=0&logType=stderr)	2018/06/17 23:14:19	20.6 h	18 ms	

Figura 9: Detalle ejecución distribuida

5. Plan de trabajo y presupuesto.

En este apartado se detalla la planificación que ha seguido el proyecto y cómo se ha organizado mediante un diagrama de Gantt, en el que se detallan las tareas realizadas.

Posteriormente, se muestra y explica el presupuesto total del proyecto en el que se tienen en cuenta las tareas realizadas, el rol de los participantes en el proyecto y los cálculos correspondientes para cubrir la aplicación de este estudio.

5.1 Diagrama de Gantt

Una forma eficaz de gestionar un proyecto es usar un diagrama de Gantt. Esta herramienta proporciona varios detalles importantes, aclara los pasos principales que debe seguir el estudio, informa sobre el tiempo máximo que se puede dedicar a trabajar en cada una de las tareas y da una visión general del plazo en el que estará acabado el estudio [19].

Es fácil observar de un vistazo las acciones previstas y realizar un seguimiento de las mismas. También se reflejan las tareas cuyo desarrollo discurre de forma paralela y, de esta manera, se puede asignar a cada actividad los recursos que necesita con el objetivo de controlar costes y personal requeridos. A continuación, podemos observar el diagrama de Gantt que ha seguido este proyecto.

Este proyecto tuvo su comienzo el 21 de septiembre de 2017 con la primera reunión realizada con el tutor. Y su fecha fin ha sido el 19 de junio de 2018, momento en el que se entrega la memoria.

En la siguiente imagen se muestra la duración de cada una de las tareas:

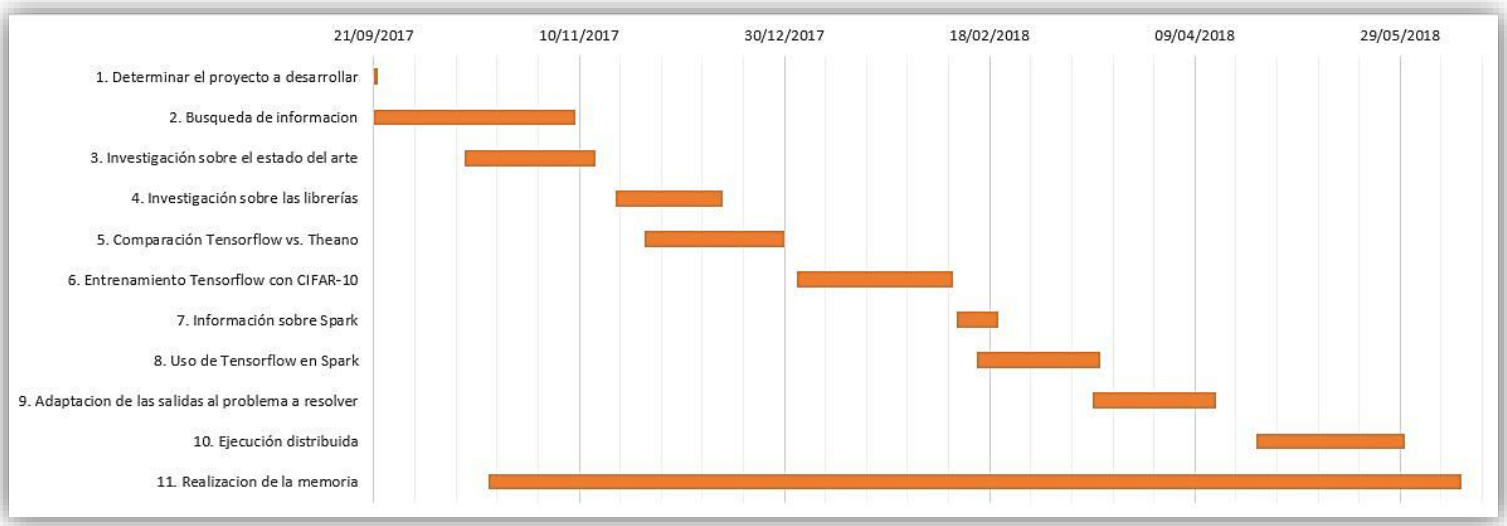


Figura 10: Diagrama de Gantt

5.2 Presupuesto

Resulta necesario conocer el coste que implicaría llevar a cabo el proyecto, por lo que se procede a realizar su cálculo en base al plan de trabajo previsto. En este presupuesto hay que tener en cuenta los costes de personal, de software, de hardware y posibles gastos extras que sean necesarios.

5.2.1 Costes de personal

En cuanto a personal, se estima que son necesarios dos perfiles.

- Ingeniero Senior (Tutor). Ingeniero con experiencia en el sector *Big Data*. El cual se encargará de supervisar el avance del proyecto y la consecución de los objetivos.
- Ingeniero Junior (Alumna). Persona encargada del desarrollo del proyecto en el que se incluyen tareas como la documentación, desarrollo del programa y la realización de las pruebas necesarias para la evaluación de dicho programa.

A continuación, se realiza el cálculo del coste total del trabajo desarrollado por el personal en función de su experiencia y del número de horas dedicadas. En la siguiente tabla se detallan estos números.

Empleado	Total horas	Coste por hora	Coste total
Ingeniero Senior	82	21,45 €/hora	1.758,9 €
Ingeniero Junior	413	13 €/hora	5.369 €
	TOTAL		**7.127,9 €**

Figura 11: Coste de personal

Los costes han sido obtenidos según lo establecido por el BOE [20]. Se han realizado los cálculos pertinentes para conseguir el coste por hora y, de esa manera, calcular el cómputo general.

En concreto, el coste del ingeniero senior se ha calculado a partir del obtenido para el ingeniero junior pero multiplicándolo por un factor del 50 % y añadiendo un 10 % adicional por revisiones extra que hayan sido necesarias.

5.2.2 Costes de herramientas

Hay que tener en cuenta también los recursos utilizados para llevar a cabo este proyecto. Por este motivo, estimamos tanto el gasto del equipo informático como de las herramientas software.

El equipo hardware utilizado genera un gasto que hay que tener en cuenta según el tiempo que este haya sido usado. Para poder realizar este cálculo consecuentemente, se estima que la vida útil del equipo es de 3 años (36 meses). En la siguiente tabla se detalla el gasto generado.

Producto	Precio	Coste/Tiempo	Tiempo de uso	Precio total
Ordenador	1.500 €	41,67 €/mes	10 meses	416,7 €
4 granjas de servidores	64000€	1,46 €/hora	200 horas	292 €
		TOTAL		**708,70 €**

Figura 12: Coste hardware

Cabe destacar que, si en lugar de haber usado las granjas de servidores con características de 64 Gb de memoria y 16 núcleos para ejecución, se hubieran alquilado los servicios a AWS (*Amazon Web Services*) de manera que tengan las mismas características, los gastos hubieran ascendido a 3704 €. Este precio se ha obtenido mediante la calculadora que ofrece su propia página web [21].

Ya que en este proyecto se han usado librerías *open source,* no supone gastos adicionales que haya que tener en cuenta. Es decir, las herramientas software suman un gasto de 0 €, entre las que se incluyen: *Anaconda Navigator* y *Tensorflow*. Otras herramientas adicionales de edición de textos (Microsoft Office) no suponen gastos tampoco pues han sido provistas por la universidad a través de convenio.

5.2.3 Costes añadidos

Hay que incluir otro tipo de gastos presentes durante el proyecto y que son adicionales a los calculados anteriormente. Se van a tener en cuenta los gastos de material de oficina y el acceso a internet. En la siguiente tabla se detallan estos números.

Producto	Precio total
Material de oficina	20 €
Conexión a internet	100 €
TOTAL	**120 €**

Figura 13: Costes adicionales

5.2.4 Presupuesto total

Se muestran, a continuación, los gastos totales detallados en los anteriores apartados. Se obtiene un gasto total del proyecto de

CONCEPTO	COSTE TOTAL
Coste de personal	7.127,9 €
Coste de hardware	708,70 €
Coste de software	0 €
Costes adicionales	120 €
TOTAL	**7.956,6 €**
IVA (21 %)	1.670,88 €
TOTAL PRESUPUESTO (con IVA)	**9.627,5 €**

Figura 14: Coste total del proyecto

6. Marco regulador

Esta sección está dedicada a detallar el marco regulador que afecta al trabajo realizado.

6.1 Programación Python

El código desarrollado en esta práctica se ha realizado en el lenguaje Python. Este lenguaje es de orientación a objetos y posee una licencia de código abierto.

6.2 Conjunto de imágenes CIFAR-10

El conjunto de imágenes que ha sido empleado en este proyecto para hacer que el sistema aprenda, es CIFAR-10 [16] [17]. Esta base de datos de imágenes es muy utilizada en aquellos procesos de reconocimiento de objetos. La licencia de uso está definida bajo la licencia de MIT [22].

6.3 Spark, Tensorflow, Tensorflow on Spark

Tanto *Spark* como plataforma, como *Tensorflow* y *Tensorflow on Spark* como librerías han sido usadas en este estudio para gestionar la red neuronal. Las tres se encuentran bajo la licencia de Apache versión 2.0, de Enero 2014 [23].

7. Impacto socio-económico

En esta sección se recogen aquellas consecuencias que pueden derivar de la implantación del sistema de seguridad creado en este proyecto.

- Mejora de la seguridad de las zonas con riesgo de ataques.
- Reducción del número de empleados de seguridad nacional que trabajaban hasta el momento para esta misma función, siendo posible así la redistribución de estos cuerpos de seguridad en otras zonas pudiendo así atender otros posibles problemas.
- Tranquilidad de los viandantes por dos motivos:
    - Mejor protección de aquellas zonas con riesgo, lo que implica mayor satisfacción general.
    - Este sistema de seguridad no es apreciable a simple vista como los métodos actuales (furgones de protección, maceteros u otros objetos de blindaje) que, además de añadir un extra de seguridad, también aportan una cierta sensación de inseguridad ante su presencia.

8. Conclusiones y trabajos futuros.

Este estudio es una buena manera de usar los datos que genera la sociedad, ofrece un servicio de ayuda más a la seguridad ciudadana y, por tanto, un servicio más que puede llegar a ofrecer una Smart City.

Una Smart City [24] es aquella ciudad en la que se monitorizan, mediante sensores, distintos elementos y factores que rodean a la sociedad. De esta manera, se recogen datos que, una vez analizados, pueden proporcionar una eficiencia de recursos, mejora del servicio prestado a los ciudadanos y una evolución de la ciudad. Algunos de los logros conseguidos son:

- Reducción de coste energético mediante la regulación de la iluminación.
- Aplicaciones móviles para interactuar con la ciudad.
- Suministro inteligente del agua, gestión inteligente de la energía y gestión más eficaz de los residuos.
- Sistemas de vigilancia para maximizar la seguridad sobre el ciudadano.
- Mejora sobre el control del tráfico.
- Asistencia más rápida, desde accidentes a mejora de infraestructuras.

También cabe destacar que no todo son ventajas. El principal inconveniente para la expansión de estos servicios es la inversión que requiere por parte de la Administración. Pero ya que estas medidas inteligentes proponen medidas de ahorro, resultaría ser una inversión recuperable.

Como se ha visto en el capítulo de experimentos, se puede concluir a partir de ellos que para el problema que se está tratando en este trabajo no es conveniente el uso de *Spark*, ya que no se tratan datos en grandes cantidades. Por lo que la ejecución más adecuada es la que se realiza en una sola máquina. En el caso de que se hubiera realizado un clasificador usando gran cantidad de datos, se hubieran obtenido mejores resultados en distribuido pues se realiza un mejor balanceo de datos. Se concluye, por tanto, que las herramientas de *Big Data* no pueden aplicarse a todos los problemas.

El objetivo principal de este trabajo se centraba en detectar un tipo de vehículos en concreto en las imágenes que se proporcionaban. Este objetivo se ha cumplido con un precisión del 95 %, por lo que se puede concluir que el estudio ha sido exitoso.

Como posibles líneas de trabajo futuras que puedan mejorar esta aplicación se encuentran algunas como:

- Implementar esta misma herramienta, pero con respuesta en tiempo real. Esta supondría generar una alarma de manera aún más rápida para avisar a los cuerpos de seguridad y actuar con mayor rapidez ante un posible problema. Esto sería posible llevarlo a cabo gracias a la extensión de *Spark* llamada *Spark Streaming*, la cual permite realizar procesamiento escalable, tiene un alto rendimiento y es robusto frente a errores [25]. Funciona de manera que en primer lugar recibe un flujo de datos en tiempo real, divide los datos en paquetes que luego son procesados por el motor *Spark*, el cual genera a su salida el resultado final también por lotes. Esta herramienta está siendo cada vez más usada ya que mejora las prestaciones anteriores. Se puede observar su funcionamiento en la siguiente figura:

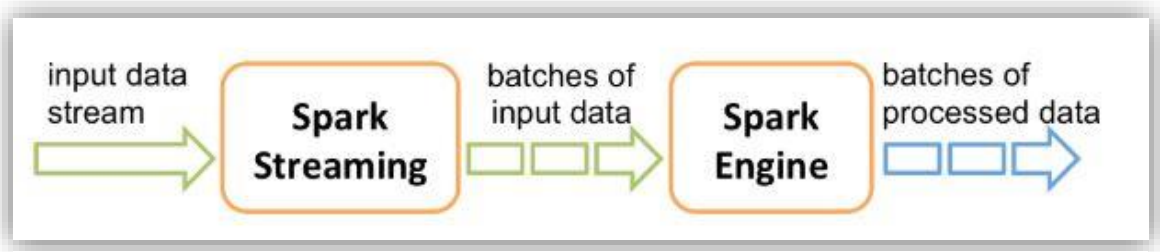


Figura 15: Funcionamiento Spark Streaming

Cabe destacar que las herramientas que integran *Deep Learning* con plataformas de *Big Data* se pueden llevar a la práctica, pero hasta el momento se encuentran en una fase temprana de desarrollo.

9. Bibliografía

[1] Matich, Damian Jorge. Redes Neuronales: Conceptos básicos y Aplicaciones. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf (Marzo 2001).[consultado el 23/05/2018]

[2] Marvin, L. Minsky y Seymour A. Papert. *Perceptrons: An introduction to Computational Geometry.* MIT Press. (1987).

[3] **Implementación de Tecnicas Deep Learning.** https://riull.ull.es/xmlui/bitstream/handle/915/1409/Implementacion%20de%20Tecnicas%20de%20Deep%20Learning.pdf?sequence=1&isAllowed=y/ [consultado el 20/04/2018]

[4] **¿Qué es y cómo funciona "Deep Learning"?**

https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/ [consultado el 20/04/2018]

[5] **Redes Neuronales.** https://ml4a.github.io/ml4a/es/neural_networks/ [consultado el 20/04/2018]

[6] **ReLU and Softmax Activation Functions**. https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions [consultado el 20/04/2018]

[7] **Inteligencia Artificial: Redes Neuronales.** https://eltamiz.com/elcedazo/2008/10/21/inteligencia-artificial-redes-neuronales/ [consultado el 20/04/2018]

[8] **Redes neuronales convolucionales con TensorFlow**. https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/ [consultado el 20/04/2018]

[9] **Una Breve Historia del Machine Learning**. http://www.synergicpartners.com/una-breve-historia-del-machine-learning/ [consultado el 20/04/2018]

[10] **Caffe**. http://caffe.berkeleyvision.org/ [consultado el 20/04/2018]

[11] **Theano**. http://deeplearning.net/software/theano/ [consultado el 20/04/2018]

[12] **Tensorflow**. https://www.tensorflow.org/ [consultado el 20/04/2018]

[13] **Hadoop.** http://hadoop.apache.org/ [consultado el 20/04/2018]

[14] **Apache Spark**. https://spark.apache.org/ [consultado el 20/04/2018]

[15] **TensorFlowOnSpark**. https://github.com/yahoo/TensorFlowOnSpark [consultado el 20/04/2018]

[16] **Conjunto de datos CIFAR-10** https://www.cs.toronto.edu/~kriz/cifar.html [consultado el 01/05/2018]

[17] Krizhevsky, Alex. Learning Multiple Layers of Feature from Tiny Images http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (Abril 2009) [consultado el 01/05/2018]

[18] **Ley de Amdahl** http://euler.mat.uson.mx/~havillam/ca/Slides/07%20Amdahl%20Law.pdf [consultado el 23/05/2018]

[19] **¿Qué es un diagrama de Gantt y para qué sirve?** https://www.obs-edu.com/es/blog-project-management/diagramas-de-gantt/que-es-un-diagrama-de-gantt-y-para-que-sirve [consultado el 20/04/2018]

[20] **Disposición 542 del BOE núm. 15 de 2017** https://www.ccoo-servicios.es/archivos/ingenierias/BOE-XVIII-Convenio-ingenierias.pdf [consultado el 12/05/2018]

[21] **Amazon Web Services Simple Monthly Calculator** https://calculator.s3.amazonaws.com/index.html [consultado el 12/05/2018]

[22] **The MIT License** https://opensource.org/licenses/MIT [consultado el 22/05/2018]

[23] **The Apache Software Foundation. Licenses** https://www.apache.org/licenses/ [consultado el 12/05/2018]

[24] **¿Qué son las Smart city o ciudades inteligentes?** http://economipedia.com/definiciones/que-es-una-ciudad-inteligente-o-smart-city.html [consultado el 20/04/2018]

[25] **Spark Streaming** https://spark.apache.org/streaming/ [consultado el 10/06/2018]